Spring Cacheでパフォーマンスを向上させよう

第1.0版　2025年04月XX日

目次

1.今回の発表の目的

今回の発表の目的

- Spring Cacheの学習で学んだことを共有する

- 業務でパフォーマンスの問題に直面した時に
  役立てていただく

2.キャッシュとは

キャッシュとは

- データや計算結果を一時的に保存する仕組みのこと
- 再度同じデータを使用する時に計算処理やデータベース・APIなどの外部参照を省略し、**パフォーマンスを向上させる**
- 一般的にメモリ上に保存されるため、**高速にデータへアクセスできる**

3. Spring Cache

Spring Cacheとは

- Spring Frameworkが提供するキャッシュ機能

- Spring AOP を使用したSpring Cache Abstractionを採用しており、
  キャッシュ実装を抽象化することで柔軟な設計を可能としている

基本概念

- キャッシュ: データを保存しておくストレージ領域
- キャッシュキー: キャッシュに保存されているデータを識別するためのキー
- キャッシュマネージャー: キャッシュの作成、取得、削除などの
  操作を管理する仕組み
- キャッシュリゾルバー: アノテーションで行われるキャッシュ操作を
  動的に解決する仕組み
- キージェネレーター: キャッシュキーをカスタマイズできる仕組み

- キャッシュライブラリ: 抽象化したキャッシュ機能の実装として利用される
  キャッシュの管理を担う外部ライブラリ

- 他にも、キャッシュの同期化などの機能も

キャッシュライブラリについて

- サポートされているキャッシュライブラリは以下の順番で検出しようとする
  - ライブラリ仕様に準じた設定があれば、そのライブラリが使用される
    1. 汎用
    2. JCache (JSR-107) (EhCache 3、Hazelcast、Infinispan など)
    3. Hazelcast
    4. Infinispan
    5. Couchbase
    6. Redis
    7. Caffeine
    8. Cache2k
    9. シンプル

使い方

アノテーションと設定ファイル(application.yml)を用いてキャッシュ機能を実装する

- @Cacheable: キャッシュ作成をトリガーする

```
@Component
@Slf4j
public class FractalService {

    @Cacheable(cacheNames = "fractal-crate-cahe" , key = "#size", sync = true)
    public BufferedImage cacheCreateFractal(Size size) {
        GenerateFractal fractal = new GenerateFractal();

        log.info("キャッシュありのフラクタル画像生成です。size: {}", size.toString());
        return fractal.generateFractal(size);
    }

}
```

- @CacheEvict: キャッシュエビクションをトリガーし、キャッシュを削除する

```
@Service
public class CacheEvictService {

    @CacheEvict(cacheNames = "cache-controll", key = "#str")
    public void evict(String str) {
    }

}
```

- @CachePut: メソッドの実行を妨げることなくキャッシュを更新する

```
@Service
@RequiredArgsConstructor
public class CachePutService {

    private final HeavyTaskLogic heavyTaskLogic;

    @CachePut(cacheNames = "cache-controll", key = "#str")
    public String put(String str, String newStr) {
        heavyTaskLogic.heavyTask();

        return "Arg:" + newStr;
    }

}
```

- @CacheConfig: クラスレベルでいくつかの一般的なキャッシュ関連の設定を共有できる

```
@CacheConfig(
        cacheNames = "fractal-cahe",
        keyGenerator = "fractalKeyGenerater",
        cacheResolver = "fractalCacheResolver")
public class CacheFractalService {
```

- @EnableCaching: キャッシングアノテーションの有効化

```
@SpringBootApplication
@EnableCaching
public class SpringCacheTestX1Application {

    public static void main(String[] args) {
        SpringApplication.run(SpringCacheTestX1Application.class, args);
    }

}
```

- application.yml

```yaml
spring:
  application:
    name: spring-cache-test-x1
  cache:
    cache-names: "fractal-crate-cahe"
    caffeine:
      spec: "maximumSize=500,expireAfterAccess=10m"
```

※他にも様々な使い方があります。

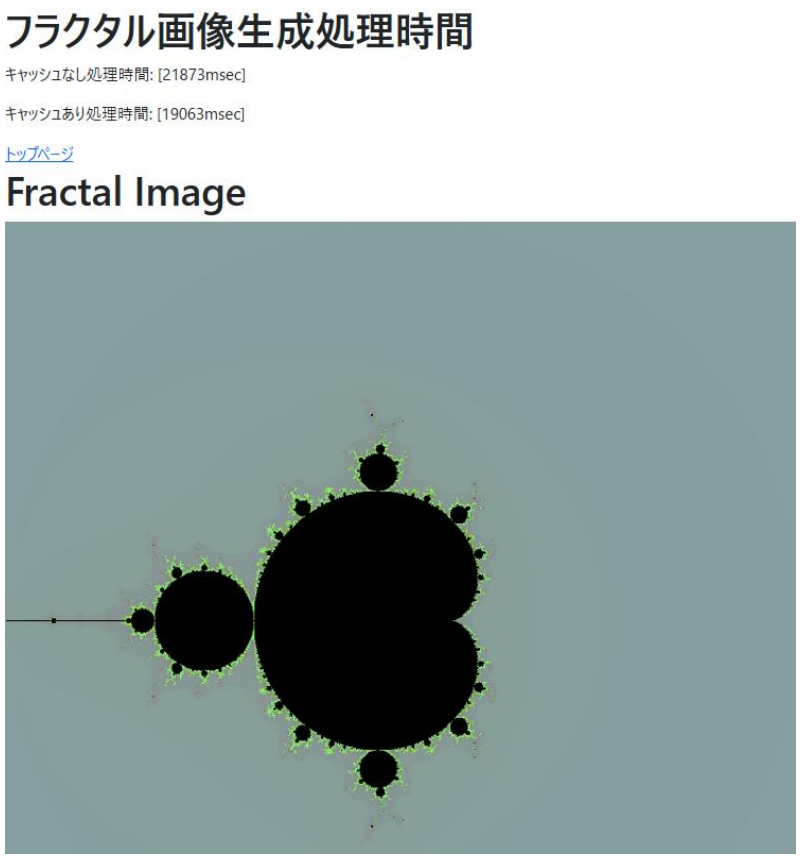
4.使用例

アプリの紹介

- フラクタル画像を生成する
- サービス側でフラクタル画像を生成、画面側で画像と処理時間を表示する
- 画面にはキャッシュ無しの時の処理時間とキャッシュ有りの時の
  処理時間が表示される
- ソース
  - https://github.com/kohei-miyadai07/SpringCacheTest/tree/master/spring-cache-test-x1

実行結果

- 初回生成（キャッシュ無し）
  - キャッシュなし処理時間: [21873msec]
  - キャッシュあり処理時間: [19063msec]



- 2回目生成（キャッシュ有り）
  - キャッシュなし処理時間: [21904msec]
  - キャッシュあり処理時間: [7msec]

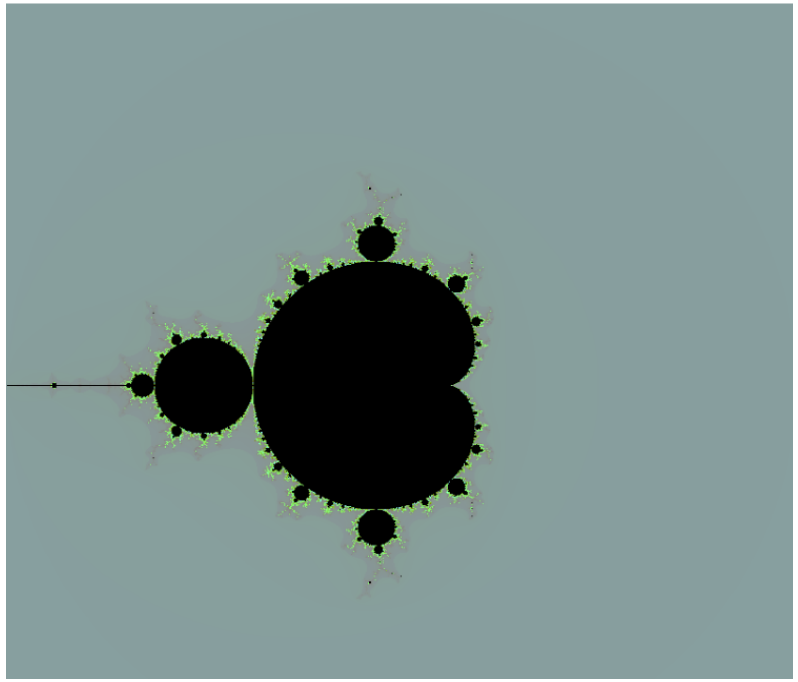
フラクタル画像生成処理時間

キャッシュなし処理時間: [21904msec]

キャッシュあり処理時間: [7msec]

トップページ

Fractal Image



5.実装の上で気を付けること

- 「キャッシュの中身」をアプリ内で変化するオブジェクトなどにすると、キャッシュの更新・削除処理の実装が必要になり、設計によっては複雑化することもある
  - **パフォーマンスに影響する可能性もある**

- キャッシュキーも同じで、アプリで操作されるオブジェクトを指定すると、キャッシュの操作をしたいときに、キーの状態を考えて実装しなくてはならなくなり、実装が複雑化する

- 今思う簡単な実装は、「キー自体をアプリで変更されるものにして、
  キーが変わるたびに中身が再取得されるようにする」こと

- SpringAOPの仕組みにより、プロシキを介して動作するため、
  自クラス内で呼び出すと、キャッシュが有効化しない

- Springの仕組みで動作するので、キャッシュの呼び出しなどはSpringの仕組みを利用する必要がある

- **他にも気を付けるべき箇所があれば、共有お願い致します**

6.資料

- SpringのCache機能をサクッと利用する
- キャッシング :: Spring Boot - リファレンスドキュメント
- Spring Bootにおけるキャッシュの有効化と利用方法 - ぽんこつ日記
- Spring Cache による処理高速化の実現
- キャッシュの抽象化について :: Spring Framework - リファレンス
- Spring BootでSpring Cache(Cache Abstraction)を試す - abcdefg.....

EOF